Mining Apps to Learn Normal Behavior

Andreas Zeller
Saarland University, Saarbrücken, Germany

Joint work with Alessandra Gorla, Ilaria Tavecchia, Vitalii Avdiienko, Konstantin Kuznetsov, and Florian Gross
Saarbrücken
Saarbrücken

1700 BSc + MSc students
375 PhD students
200 Researchers (post PhD)
8 New buildings since 2001

7 ERC Grant holders
6 Leibniz Awardees
4 ACM Fellows
1 Software Engineer
Specifications

removeChild

\[ \Delta \text{XMLElement} \]

\text{child? : XML\_ELEMENT} \\
\text{child?} \in \text{enumerateChildren} \\
\text{child?} \neq \text{null} \\
\text{enumerateChildren}' = \text{enumerateChildren} \setminus \text{child?} \\
\text{getChildrenCount}' = \text{getChildrenCount} - 1
This is precisely what our proposed approach produces: Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

```
removeChild(child?
XMLElement child?
\rightarrow
\text{enumerateChildren}

\text{child?} \in \text{enumerateChildren}
\text{child?} \neq \text{null}
\text{enumerateChildren}' = \text{enumerateChildren} \setminus \text{child?}
\text{getChildrenCount}' = \text{getChildrenCount} - 1
```

Note how the specification captures two important preconditions not stated in the documentation—

- `child` be a child of the target node,
- `child` be non-null.

Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

### 1d.3 State of the Art

### 1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The `removeChild()` code indeed reveals some insights: From this code, any static analysis can easily deduce precondition, child? \rightarrow null.

But how would
Specifying Correctness

public class XMLElement implements IXMLElement, Serializable {
    // The name.
    private String name;

    // The child elements.
    private Vector children;

    // Returns an enumeration of all child elements.
    public Enumeration enumerateChildren() {
        ...
    }

    // Returns the number of children.
    public int getChildrenCount() {
        ...
    }

    // Removes a child element.
    public void removeChild(XMLElement child) {
        ...
    }

    // more methods and attributes...
}

Figure 1: The XMLElement class from the NanoXML parser

This is precisely what our proposed approach produces: Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild
XMLElement child?: XML_ELEMENT
child? ⇨ enumerateChildren
child? ⇨ = null
enumerateChildren = enumerateChildren \ child?
getChildrenCount = getChildrenCount − 1

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—child be a child of the target node and child be non-null. Both properties are essential for generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The removeChild() code indeed reveals some insights: From this code, any static analysis can easily deduce preconditions: child? ⇨ = null. But how would
Unknown error

Was this information helpful?
Normality
Mining Normality

This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language the mined specification for removeChild() is shown in Figure:

```
XMLElement child
XMLElement child ⇴ enumerateChildren
XMLElement child ⇴ = null

enumerateChildren = enumerateChildren \ child
getChildrenCount = getChildrenCount – 1
```

Note how the specification captures two important preconditions not stated in the documentation—that `child` be a child of the target node and that `child` be non-null. Both properties are essential for generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art

1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

```
child? ⇴ = null
```

But how would
Outliers
Looking for a restaurant, a bar, a pub or just to have fun in London? Search no more! This application has all the information you need:

- You can search for every type of food you want: french, british, chinese, indian etc.
- You can use it if you are in a car, on a bicycle or walking
- You can view all objectives on the map
- You can search objectives
- You can view objectives near you
- You can view directions (visual route, distance and duration)
- You can use it with Street View
- You can use it with Navigation

Keywords: london, restaurants, bars, pubs, food, breakfast, lunch, dinner, meal, eat, supper, street view, navigation

Also sends out account info
Also sends out mobile phone number
Also sends out your device ID
What is malicious?

- Also sends out account info
- Also sends out mobile phone number
- Also sends out your device ID

London Restaurants

WhatsApp messenger

- Also sends out account info
- Also sends out mobile phone number
- Also sends out your device ID
What is normal?

- “London Restaurants” is a “travel” app
- For “travel” apps, sending account infos is abnormal
- For “messaging” apps, this is far more likely
CHABADA

1. App collection

2. Topics

3. Clusters

"Weather", "Map"...
"Travel", "Map"...
"Theme"

Weather + Travel
Themes
1. INTRODUCTION

How do we know a program does what it claims to do? After clustering apps, we run the risk of the app being “malware”—that is, to act against the interests of its users.

In the mobile world, for any specification on what makes behavior beneficial or malicious, behavior considered malicious in one app may well be a very much depends on the current context.

Starting from a collection of “good” apps (1), we identify their description topics (2) to form clusters of related apps (3). For each cluster, we identify the APIs used, grouped by related permissions (4), and can then identify outliers that use APIs that are uncommon for that cluster (5).

Specifically, our CHABADA approach1 takes five steps, illustrated in Figure 1 and detailed later in the paper:

1. CHABADA starts with a collection of 22,500+ “good” Android applications downloaded from the Google Play Store.
2. We extract the APIs (1), and we also advertise the APIs (2), in all the examples above, the user would be informed and asked whether the behavior of the app matches the advertised.
3. Given a set of “good” APIs (3), we use the set of Android application programming interfaces (APIs) that are used from within the app binary. The key idea is to associate descriptions and APIs against advertised behavior of an app, we use a proxy for the advertised behavior of an app, we use its natural language description from the Google Play Store. As a proxy for the advertised behavior of an app, we use its natural language description from the Google Play Store.
4. We run the risk of the app being “malware”—that is, to act against the interests of its users.

1. CHABADA stands for CHecking App Behavior Against Descriptions.
CHABADA

1. App collection

2. Topics

"Weather", "Map"…

"Travel", "Map"…

"Theme"

3. Clusters

Weather + Travel

Themes

4. APIs

Internet  Access-Location

5. Outliers

Internet  Access-Location  Send-SMS
App Collection

- Source: Google Play Store
- Downloaded top 150 apps + metadata from each of the 30 categories
- Time frame: Winter to Spring 2013
- Total: 32,136 apps
- Data package available on Web site
looking for a restaurant, a bar, a pub or just to have fun in London? search no more! this application has all the information you need:
• you can search for every type of food you want: french, british, chinese, indian etc.
• you can use it if you are in a car, on a bicycle or walking
• you can view all objectives on the map
• you can search objectives
• you can view objectives near you
• you can view directions (visual route, distance and duration)
• you can use it with street view
• you can use it with navigation
keywords: London, restaurants, bars, pubs, food, breakfast, lunch, dinner, meal, eat, supper, street view, navigation
Stemming

looking for a restaurant, a bar, a pub or just to have fun in London? search no more! this application has all the information you need:

• you can search for every type of food you want: French, British, Chinese, Indian etc.
• you can use it if you are in a car, on a bicycle or walking
• you can view objectives on the map
• you can search objectives
• you can view objectives near you
• you can view directions (visual route, distance and duration)
• you can use it with street view
• you can use it with navigation

keywords: London, restaurants, bars, pubs, food, breakfast, lunch, dinner, meal, eat, supper, street view, navigation
Stemming

look london restaur search bar pub just applic fun
inform can search need everi type food want french
british chines indian etc car bicycl walk
can us can view object map visual rout
can search object search can view distanc
durat can view direct object near
can us street view can us navig
keyword london restaur bar pub food view
breakfast lunch dinner meal eat supper street navig
Topic Analysis

• Eliminated all apps with \( \leq 10 \) words, now 22,521 apps

• Want to discover the *topics* that occur in a collection of unlabeled text

• A *topic* consists of a cluster of words that frequently occur together

• Used *Latent Dirichlet Allocation* (LDA) to identify 30 topics
# Topics

<table>
<thead>
<tr>
<th>Id</th>
<th>Assigned Name</th>
<th>Most Representative Words (stemmed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>“personalize”</td>
<td>galaxi, nexu, device, screen, effect, instal, customis</td>
</tr>
<tr>
<td>1</td>
<td>“game and cheat sheets”</td>
<td>game, video, page, cheat, link, tip, trick</td>
</tr>
<tr>
<td>2</td>
<td>“money”</td>
<td>slot, machine, money, poker, currenc, market, trade, stock, casino coin, finance</td>
</tr>
<tr>
<td>3</td>
<td>“tv”</td>
<td>tv, channel, countri, live, watch, germani, nation, bbc, newspap</td>
</tr>
<tr>
<td>4</td>
<td>“music”</td>
<td>music, song, radio, play, player, listen</td>
</tr>
<tr>
<td>5</td>
<td>“holidays” and religion</td>
<td>christmas, halloween, santa, year, holiday, islam, god</td>
</tr>
<tr>
<td>6</td>
<td>“navigation and travel”</td>
<td>map, inform, track, gps, navig, travel</td>
</tr>
<tr>
<td>7</td>
<td>“language”</td>
<td>language, word, english, learn, german, translat</td>
</tr>
<tr>
<td>8</td>
<td>“share”</td>
<td>email, ad, support, facebook, share, twitter, rate, suggest</td>
</tr>
<tr>
<td>9</td>
<td>“weather and stars”</td>
<td>weather, forecast, locate, temperatur, map, city, light</td>
</tr>
<tr>
<td>10</td>
<td>“files and video”</td>
<td>file, download, video, media, support, manage, share, view, search</td>
</tr>
</tbody>
</table>

## 2.4 Clustering Apps with K-means

As an example, Table 2 shows four applications clustered with the corresponding probabilities of belonging to topics. If we applied K-means to partition the set of applications into two clusters (potentially even one per app), and we use the probabilities of belonging to topics as features with a certain probability. What we need, though, is to identify one cluster with the corresponding probabilities of belonging to topics. If we used the K-means algorithm for applications, one of the most common clustering algorithms, for each solution using K = 2..12, we compute the average of the elements' silhouette for each solution except for K = 12, as the number of clusters, and we select the best solution, i.e. the best number of clusters, according to the average silhouette of the elements. The average silhouette of the elements can be interpreted as the degree to which the elements are part of their cluster. If the value is close to 1, it means that the element is in the appropriate cluster. If the value is close to -1, it means that the element is in the wrong cluster. Thus, to identify the best solution, we compute the average of the elements' silhouette for each solution. The silhouette of an element is close to 1, it means that the element is the measure of how closely the element is matched to other elements within its cluster and how loosely it is matched to the other elements in the other clusters. The average silhouette of the elements is the measure of how closely the elements are matched to the other elements within their cluster and how loosely they are matched to the other elements in the other clusters. The average silhouette of the elements is the measure of how closely the elements are matched to the other elements within their cluster and how loosely they are matched to the other elements in the other clusters.

## 2.5 Finding the Best Number of Clusters

As with most scientific work, the approach presented in this paper would lead to some refinements. We briefly list the most important ones here as to have future researchers avoid some of the problems we encountered. The best solution, i.e. the best number of clusters, we now can search for outliers with respect to their behavior. As mentioned in Section 2.4, we fixed the number of clusterings we would then be able to evaluate. The range for the number of clusters, K, is from 2 to 12, as discussed in [19]. The silhouette of an element is close to 1, it means that the element is the measure of how closely the element is matched to other elements within its cluster and how loosely it is matched to the other elements in the other clusters. The average silhouette of the elements is the measure of how closely the elements are matched to the other elements within their cluster and how loosely they are matched to the other elements in the other clusters. The average silhouette of the elements is the measure of how closely the elements are matched to the other elements within their cluster and how loosely they are matched to the other elements in the other clusters. The average silhouette of the elements is the measure of how closely the elements are matched to the other elements within their cluster and how loosely they are matched to the other elements in the other clusters.
<table>
<thead>
<tr>
<th>Id</th>
<th>Topic</th>
<th>Most Representative Words (stemmed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>“design and art”</td>
<td>life, peopl, natur, form, feel, learn, art, design, uniqu, effect, modern</td>
</tr>
<tr>
<td>14</td>
<td>“food and recipes”</td>
<td>recip, cake, chicken, cook, food</td>
</tr>
<tr>
<td>15</td>
<td>“personalize”</td>
<td>theme, launcher, download, install, icon, menu</td>
</tr>
<tr>
<td>16</td>
<td>“health”</td>
<td>weight, bodi, exercise, diet, workout, medic</td>
</tr>
<tr>
<td>17</td>
<td>“travel”</td>
<td>citi, guid, map, travel, flag, countri, attract</td>
</tr>
<tr>
<td>18</td>
<td>“kids and bodies”</td>
<td>kid, anim, color, girl, babi, pictur, fun, draw, design, learn</td>
</tr>
<tr>
<td>19</td>
<td>“ringtones and sound”</td>
<td>sound, rington, alarm, notif, music</td>
</tr>
<tr>
<td>20</td>
<td>“game”</td>
<td>game, plai, graphic, fun, jump, level, ball, 3d, score</td>
</tr>
<tr>
<td>21</td>
<td>“search and browse”</td>
<td>search, icon, delet, bookmark, link, homepage, shortcut, browser</td>
</tr>
<tr>
<td>22</td>
<td>“battle games”</td>
<td>story, game, monster, zombi, war, battle</td>
</tr>
<tr>
<td>23</td>
<td>“settings and utils”</td>
<td>screen, set, widget, phone, batteri</td>
</tr>
<tr>
<td>24</td>
<td>“sports”</td>
<td>team, football, leagu, player, sport, basketbal</td>
</tr>
<tr>
<td>25</td>
<td>“wallpapers”</td>
<td>wallpap, live, home, screen, background, menu</td>
</tr>
<tr>
<td>26</td>
<td>“connection”</td>
<td>device, connect, network, wifi, blootooth, internet, remot, server</td>
</tr>
<tr>
<td>27</td>
<td>“policies and ads”</td>
<td>live, ad, home, applovin, notif, data, polici, pri-vacy, share, airpush, advertis</td>
</tr>
<tr>
<td>28</td>
<td>“popular media”</td>
<td>seri, video, film, album, movi, music, award, star, fan, show, gangnam, top, bieber</td>
</tr>
<tr>
<td>29</td>
<td>“puzzle and card games”</td>
<td>game, plai, level, puzzl, player, score, chal-leng, card</td>
</tr>
</tbody>
</table>
London Restaurant Topics

look london restaur search bar pub just applic fun inform can search need everi type food want french british chines indian etc car bicycl walk can us can view object map visual rout can search object search can view distanc durat can view direct object near can us street view can us navig keyword london restaur bar pub food view breakfast lunch dinner meal eat supper street navig

“navigation and travel” (59.8%)
“food and recipes” (19.9%)
“travel” (14.0%)
CHABADA

1. App collection
2. Topics
3. Clusters
4. APIs
5. Outliers
How do we know a program does what it claims to do? After clustering Android apps by their description topics, we identify outliers in each cluster with respect to their API usage. A "weather" app that sends messages thus becomes an anomaly; likewise, a "messaging" app would typically not be expected to access the current location. Applied on a set of 22,500+ Android applications, our CHABADA prototype identified several anomalies; additionally, it flagged 56% of novel malware as such, without requiring any known malware patterns.

1. INTRODUCTION

Checking whether a program does what it claims to do is a long-standing problem for developers. Unfortunately, it now has become a problem for computer users, too. Whenever we install a new app, we run the risk of the app being "malware"—that is, to act against the interests of its users.

Research and industry so far have focused on detecting malware by checking static code and dynamic behavior against predefined patterns of malicious behavior. However, this will not help against new attacks, as it is hard to define in advance whether some program behavior will be beneficial or malicious. The problem is that any specification on what makes behavior beneficial or malicious very much depends on the current context.

In the mobile world, for instance, behavior considered malicious in one app may well be a feature of another app:

• An app that sends a text message to a premium number to raise money is suspicious? Maybe, but on Android, this is a legitimate payment method for unlocking game features.

• An app that tracks your current position is malicious? Not if it is a navigation app, a trail tracker, or a map application.

• An application that takes all of your contacts and sends them to some server is malicious? This is what WhatsApp does upon initialization, one of the world's most popular mobile messaging applications.

The question thus is not whether the behavior of an app matches a specific pattern or not; it is whether the program behaves as advertised. In all the examples above, the user would be informed and asked for authorization before any questionable behavior. It is the covert behavior that is questionable or downright malicious.

In this paper, we attempt to check implemented app behavior against advertised app behavior. Our domain is Android apps, so chosen because of its market share and history of attacks and frauds. As a proxy for the advertised behavior of an app, we use its natural language description from the Google Play Store. As a proxy for its implemented behavior, we use the set of Android application programming interfaces (APIs) that are used from within the app binary. The key idea is to associate descriptions and API usage to detect anomalies: "This 'weather' application accesses the messaging API, which is unusual for this category."

Specifically, our CHABADA approach takes five steps, illustrated in Figure 1 and detailed later in the paper:

1. CHABADA starts with a collection of 22,500+ "good" Android applications downloaded from the Google Play Store.

CHABADA stands for Checking App Behavior Against Descriptions of Apps. "Chabada" is a French word for the base ternary rhythm pattern in Jazz.
Clustering

• Want to identify *groups of applications* that are similar according to their descriptions.
• Used *K-Means* to identify such clusters
• Used *elements silhouette* to identify best number K of clusters
## Clusters

<table>
<thead>
<tr>
<th>Id</th>
<th>Assigned Name</th>
<th>Size</th>
<th>Most Important Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“sharing”</td>
<td>1,453</td>
<td>share (53%), settings and utils, navigation and travel</td>
</tr>
<tr>
<td>2</td>
<td>“puzzle and card games”</td>
<td>953</td>
<td>puzzle and card games (78%), share, game</td>
</tr>
<tr>
<td>3</td>
<td>“memory puzzles”</td>
<td>1,069</td>
<td>puzzle and card games (40%), game (12%), share</td>
</tr>
<tr>
<td>4</td>
<td>“music”</td>
<td>714</td>
<td>music (58%), share, settings and utils</td>
</tr>
<tr>
<td>5</td>
<td>“music videos”</td>
<td>773</td>
<td>popular media (44%), holidays and religion (20%), share</td>
</tr>
<tr>
<td>6</td>
<td>“religious wallpapers”</td>
<td>367</td>
<td>holidays and religion (56%), design and art, wallpapers</td>
</tr>
<tr>
<td>7</td>
<td>“language”</td>
<td>602</td>
<td>language (67%), share, settings and utils</td>
</tr>
<tr>
<td>8</td>
<td>“cheat sheets”</td>
<td>785</td>
<td>game and cheat sheets (76%), share, popular media</td>
</tr>
<tr>
<td>9</td>
<td>“utils”</td>
<td>1,300</td>
<td>settings and utils (62%), share, connection</td>
</tr>
<tr>
<td>10</td>
<td>“sports game”</td>
<td>1,306</td>
<td>game (63%), battle games, puzzle and card games</td>
</tr>
<tr>
<td>11</td>
<td>“battle games”</td>
<td>953</td>
<td>battle games (60%), game</td>
</tr>
<tr>
<td>Cluster</td>
<td>Topic</td>
<td>Size</td>
<td>Most Important Topics</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------</td>
<td>------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>19</td>
<td>“sports”</td>
<td>580</td>
<td><strong>sports</strong> (62%), share, popular media</td>
</tr>
<tr>
<td>20</td>
<td>“files and videos”</td>
<td>679</td>
<td><strong>files and videos</strong> (63%), share, settings and utils</td>
</tr>
<tr>
<td>21</td>
<td>“search and browse”</td>
<td>363</td>
<td><strong>search and browse</strong> (64%), game, puzzle and card games</td>
</tr>
<tr>
<td>22</td>
<td>“advertisements”</td>
<td>380</td>
<td><strong>policies and ads</strong> (97%)</td>
</tr>
<tr>
<td>23</td>
<td>“design and art”</td>
<td>978</td>
<td><strong>design and art</strong> (48%), share, game</td>
</tr>
<tr>
<td>24</td>
<td>“car games”</td>
<td>449</td>
<td><strong>cars</strong> (51%), game, puzzle and card games</td>
</tr>
<tr>
<td>25</td>
<td>“tv live”</td>
<td>500</td>
<td><strong>tv</strong> (57%), share, navigation and travel</td>
</tr>
<tr>
<td>26</td>
<td>“adult photo”</td>
<td>828</td>
<td><strong>photo and social</strong> (59%), share, settings and utils</td>
</tr>
<tr>
<td>27</td>
<td>“adult wallpapers”</td>
<td>543</td>
<td><strong>wallpapers</strong> (51%), share, kids and bodies</td>
</tr>
<tr>
<td>28</td>
<td>“ad wallpapers”</td>
<td>180</td>
<td><strong>policies and ads</strong> (46%), wallpapers, settings and utils</td>
</tr>
<tr>
<td>29</td>
<td>“ringtones and sound”</td>
<td>662</td>
<td><strong>ringtones and sound</strong> (68%), share, settings and utils</td>
</tr>
<tr>
<td>30</td>
<td>“theme wallpapers”</td>
<td>593</td>
<td><strong>wallpapers</strong> (90%), holidays and religion, share</td>
</tr>
<tr>
<td>31</td>
<td>“personalize”</td>
<td>402</td>
<td><strong>personalize</strong> (86%), share, settings and utils</td>
</tr>
<tr>
<td>32</td>
<td>“settings and wallpapers”</td>
<td>251</td>
<td><strong>settings and utils</strong> (37%), <strong>wallpapers</strong> (37%), personalize</td>
</tr>
</tbody>
</table>
“Personalize” Cluster
“Travel” Cluster
CHABADA

1. App collection

2. Topics

3. Clusters

4. APIs

5. Outliers

Starting from a collection of "good" Android applications downloaded from the Google Play Store, we identify their description topics to form clusters of related apps. For each cluster, we identify the APIs used, grouped by related permissions, and can then identify outliers that use APIs that are uncommon for that cluster.

The question thus is not whether the behavior of an app matches a specific pattern or not; it is whether the program behaves as advertised. In all the examples above, the user would be informed and asked for authorization before any questionable behavior. It is the covert behavior that is questionable or downright malicious.

In this paper, we attempt to check implemented app behavior against advertised app behavior. Our domain is Android apps, so chosen because of its market share and history of attacks and frauds. As a proxy for the advertised behavior of an app, we use its natural language description from the Google Play Store. As a proxy for its implemented behavior, we use the set of Android application programming interfaces (APIs) that are used from within the app binary. The key idea is to associate descriptions and API usage to detect anomalies: "This 'weather' application accesses the messaging API, which is unusual for this category."

Specifically, our CHABADA approach takes five steps, illustrated in Figure 1 and detailed later in the paper:

1. CHABADA starts with a collection of 22,500+ "good" Android applications downloaded from the Google Play Store.

CHABADA stands for CHecking App Behavior Against Descriptions of Apps. "Chabada" is a French word for the base ternary rhythm pattern in Jazz.
API Analysis

• For each APK, we identified the APIs used
• Used simple static analysis
• Only considered *sensitive APIs* which would be governed by *permissions*
London Restaurants

android.net.ConnectivityManager.getActiveNetworkInfo()
android.webkit.WebView()

INTERNET
GET-ACCOUNTS
ACCESS-WIFI-STATE
ACCESS-NETWORK-STATE
ACCESS-FINE-LOCATION
READ-PHONE-STATE
VIBRATE

android.net.NetworkInfo.isConnectedOrConnecting()
android.net.ConnectivityManager.getAllNetworkInfo()
“Travel” Cluster

Description

Permissions of APIs used

ACCESS-FINE-LOCATION
ACCESS-NETWORK-STATE
INTERNET
READ-PHONE-STATE
VIBRATE
WRITE-EXTERNAL-STORAGE
WAKE-LOCK
“Personalize” Cluster

Description

Permissions of APIs used
1. INTRODUCTION

How do we know a program does what it claims to do? After clustering, it is common to have a set of "good" apps that have been checked. This helps in identifying new attacks, as it is hard to define in advance whether some program behavior will be beneficial or malicious. The problem is that new attacks can be introduced, and it is hard to define in advance what makes behavior beneficial or malicious. The key is to identify outliers in each cluster with respect to their API usage. A "weather" app might access the current location, which is unusual for this category.

2. Topics

An app that sends messages becomes an anomaly; likewise, a "messaging" app would typically not be expected to access the current location. Applied on a set of 22,500+ Android applications, our CHABADA prototype identified several anomalies; additionally, it flagged 56% of novel malware as such, without requiring any static code and dynamic behavior against predefined patterns of malicious behavior. However, this will not help against unknown malware patterns.

3. Clusters

In all the examples above, the user would be informed and asked for authorization before any questionable behavior. It is uncommon for that cluster (5) to access the current location. In Figure 1 and detailed later in the paper: "Weather", "Map", "Travel", "Map", "Theme".

4. APIs

Starting from a collection of "good" apps (1), we identify their API usage. Our domain is Android apps, and we use the set of Android application programming interfaces (APIs) that are used from within its natural language description from the Google Play Store. As a proxy for its implemented behavior, we use the set of Android API usage. As a proxy for the advertised behavior of an app, we use the set of Android API usage.

5. Outliers

In all the examples above, the user would be informed and asked for authorization before any questionable behavior. It is uncommon for that cluster (5) to access the current location. In Figure 1 and detailed later in the paper: "Weather", "Map", "Travel", "Map", "Theme".
CHABADA

1. App collection

2. Topics

3. Clusters

4. APIs

Internet
Access-Location

"Weather",
"Map"

"Travel",
"Map"

"Theme"

4. APIs

Internet
Access-Location

Send-SMS

5. Outliers

"Weather + Travel
Themes

"Weather",
"Map"

"Travel",
"Map"

"Theme"

"Weather",
"Map"

"Travel",
"Map"

"Theme"
“Travel” Cluster

Description

Permissions of APIs used

ACCESS-FINE-LOCATION
ACCESS-NETWORK-STATE
INTERNET
“Travel” Cluster

Permissions of APIs used

London Restaurants

Permissions of APIs used
London Restaurants

- android.net.ConnectivityManager.getActiveNetworkInfo()
- android.webkit.WebView()
- java.net.HttpURLConnection.connect()
- android.app.NotificationManager.notify()
- java.net.URL.openConnection()
- android.telephony.TelephonyManager.getDeviceId()
- android.location.LocationManager.getBestProvider()
- android.telephony.TelephonyManager.getLine1Number()
- android.net.wifi.WifiManager.isWifiEnabled()
- android.accounts.AccountManager.getAccountsByType()
- android.net.wifi.WifiManager.getConnectionInfo()
- android.location.LocationManager.getLastKnownLocation()
- android.location.LocationManager.isProviderEnabled()
- android.location.LocationManager.requestLocationUpdates()
- android.net.NetworkInfo.isConnectedOrConnecting()
- android.net.ConnectivityManager.getAllNetworkInfo()

→ An Outlier in the “Travel” Cluster
Outlier Analysis

- In each cluster, identified outliers through one-class support vector machine (OC-SVM)
- Features of each APK: a vector of (Sensitive API, #call sites)
London Restaurants

```
android.net.ConnectivityManager.getActiveNetworkInfo()
android.webkit.WebView()
java.net.HttpURLConnection.connect()
android.app.NotificationManager.notify()
java.net.URL.openConnection()
android.telephony.TelephonyManager.getDeviceId()
org.apache.http.impl.client.DefaultHttpClient.execute()
android.telephony.TelephonyManager.getLine1Number()
android.net.wifi.WifiManager.getConnectionInfo()
android.net.wifi.WifiManager.isWifiEnabled()
android.telephony.TelephonyManager.getBestProvider()
android.accounts.AccountManager.getAccountsByType()
android.net.wifi.WifiManager.getConnectionInfo()
android.location.LocationManager.getLastKnownLocation()
android.location.LocationManager.isProviderEnabled()
android.location.LocationManager.requestLocationUpdates()
android.net.NetworkInfo.isConnectedOrConnecting()
android.net.ConnectivityManager.getActiveNetworkInfo()
```

→ Identified as Outlier
In this paper, we attempt to check whether a program behaves as advertised. The question thus is not whether the behavior of an app matches its description, but whether it does what it claims to do. Our domain is Android apps, so chosen because of its market share and history of attacks and frauds. As a proxy for the advertised behavior of an app, we use its natural language description from the Google Play Store. As a result, our CHABADA prototype identified several anomalies; additionally, it flagged 56% of novel malware as such, without requiring any specific training of the model.

Starting from a collection of “good” Android applications downloaded from the Google Play Store of over 22,500+ apps, we run the risk of the app being “malware” — that is, to act against the interests of its users. This is a problem for computer users, too. Whenever we install a new app, we ask for a legitimate payment method for unlocking game features. Asking for the current location? Not if this is suspicious. As what is a navigation app, a trail tracker, or a map application. In all the examples above, the user would be informed and asked for authorization before any questionable behavior. It is the advertised behavior that is questionable or downright malicious. In the mobile world, for instance, behavior considered malicious in one app may well be a very much depends on the current context.

How do we know a program does what it claims to do? After clustering applications from the same category, we can see which apps have APIs that are very different from the rest. The key idea is to associate descriptions and APIs used from within the app binary. The key idea is to associate the described behavior with the API usage of the app. By checking static code and dynamic behavior against predefined patterns of malicious behavior, we can identify if the app is behaving as advertised or not.

Specifically, our CHABADA approach takes five steps, illustrated in Figure 1 and detailed later in the paper:

1. CHABADA starts with a collection of 22,500+ “good” Android applications downloaded from the Google Play Store.
2. We group the apps into topics based on their description topics. For instance, apps with the topic “Weather” would typically not access the current location.
3. We form clusters of related apps, grouping them by related permissions.
4. We identify the APIs used by each cluster, grouped by related permissions.
5. We identify outliers that use APIs that are uncommon for that cluster.
Evaluation: Outliers

• Can our technique effectively identify anomalies (i.e., mismatches between description and behavior) in Android apps?

• Manually checked top 5 outliers in each cluster (160 total)

• 26% showed covert behavior using sensitive APIs that acts against the interest of its users.
What makes an outlier?

- Ad frameworks (apploving, airpush)
- Dubious behavior (UNO, WICKED, Yahoo!)
- Uncommon behavior (SoundCloud)
- Benign outliers (Mr. Will’s Stud Poker)
Evaluation: Malware

- Can our technique be used to identify malicious Android applications?
- In each cluster, trained OC-SVM on 90% of “benign” apps
- Used TF-IDF as classifier on sets with remaining “benign” apps and 173 known malware apps

Malware recognition rate >80%
Information Flow

- Which sensitive APIs does the device ID flow to?
MUDFLOW

App1

App2

App3

LOG1

LOG2

ID4

ID4

ID2

SMS2

Outlier Detector

Outlier Detection

Training

Malware recognition rate >86%
Úlfar Erlingsson
Facebook ermöglicht es dir, mit den Menschen in deinem Leben in Verbindung zu treten und Inhalte mit diesen zu teilen.

Registrieren
Facebook ist und bleibt kostenlos.

Vorname:

Name:

Deine E-Mail-Adresse:

E-Mail nochmals:

Geschlecht auswählen:

Ich bin: Geschlecht auswählen:

Geburtstag: Tag: Monat: Jahr:

Warum muss ich meinen Geburtstag angeben?

Wenn du auf „Registrieren“ klickst, akzeptierst du unsere Nutzungsbedingungen und erklären unsere Datenschutzrichtlinien gelesen und verstanden zu haben.

Erstelle eine Seite für eine Berührtheit, eine Braut oder ein
App Mining

- For 100,000s of apps:
- Gather *descriptions*
- Gather *metadata*
- Gather *execution features*
- Find what is *common* and what is *uncommon*
Mining Apps to Learn Normal Behavior

Andreas Zeller
Saarland University, Saarbrücken, Germany

Joint work with Alessandra Gorla, Ilaria Tavecchia, Vitalii Avdiienko, Konstantin Kuznetsov, and Florian Gross

Key Findings

- Of the top five API outliers per cluster, 26% show unadvertised (covert) behavior
- Typically ad frameworks (apploving, airpush)
- Several anomalies (UNO, WICKED, Yahoo! Mail…)
- Using TF-IDF to classify API outliers per cluster, we could flag >80% of novel malware as such
- Current work: Dynamic API usage, information flow, user authorization

http://www.st.cs.uni-saarland.de/chabada/